MACHINE CODE MONITOR

ACCUMULATOR

DATA BUS BUFFER

6502 Assembly Language Monitor. Features include Assembly/Disassembly, Hex Dump, Relocate, Breakpoint and much more.

The information in this manual has been reviewed and is believed to be entirely reliable. No responsibility, however, is assumed for inaccuracies. The material in this manual is for information purposes only, and is subject to change without notice.

© 1982 COMMODORE INTERNATIONAL
All rights reserved. No part of this program or accompanying instruction leaflet may be duplicated, copied, transmitted or reproduced in any form or by any means without the prior written permission of Commodore Home Computer Division.

Commodore Home Computer Division
675 Ajax Avenue, Slough Trading Estate,
Slough, Berks. SL1 4BG England.

Printed in England

MACHINE CODE MONITOR (VICMON) USER MANUAL

TABLE OF CONTENTS

TABLE OF FIGURES

SECTION ONE
INTRODUCTION TO VICMON

1.1 Introduction

VICMON is the nickname of the hexadecimal machine code monitor designed to enable easy debugging of machine code programs which are resident in a VIC 20 computer system.

This manual does not set out to teach machine code programming on the VIC. Before attempting to use machine code on the VIC you should refer to the following:

MOS 6502 Programming Manuals
VIC Programmers' Reference Guide
VIC Zero Page Memory Map.

Useful reading:

6502 Assembly Language Programming
by Leventhal.

These are available from most COMMODORE Computer dealers.

VICMON and this manual are intended for use by people with some programming experience and some knowledge of 6502 CBM machine code programming, but a high level of expertise is not required.

1.2 The VICMON Manual

This manual is divided into three parts which are outlined below.

SECTION ONE — INTRODUCTION TO VICMON

This section outlines VICMON in general terms. It explains the conventions used by this manual when describing the command formats. How to start VICMON is also included.

SECTION TWO — THE COMMANDS OF VICMON

In this section, each VICMON command is explained, its format shown and an example given. The commands are in alphabetical order.

SECTION THREE — USING VICMON AS A DEBUGGING TOOL

This section uses an actual machine language program to show how VICMON can be used to locate faults in the program.

1.3 VICMON Functions

VICMON offers the following functions:

★ Displaying chosen areas of memory.
★ Changing contents of memory locations.
★ Moving blocks of memory.
★ Filling selected blocks of memory.
★ Searching memory for a pattern.
★ Examining and changing registers.
★ Setting breakpoints.
★ Executing programs with breakpoint control.
★ Storing and retrieving data and programs.
★ Executing programs at three different speed options.

1.4 Starting the VICMON System

The VICMON cartridge must always be inserted or removed from the VIC with the power off. The cartridge is inserted into the expansion port with the label on the cartridge facing up.

If a VIC 1010 Memory Expansion Board is in use, this should also be turned off. VICMON may be used in conjunction with VIC 1212 Programmers' Aid and/or VIC 1211A Super Expander cartridges. However, please note that some operations may conflict if changing from one cartridge to another. Therefore the VIC may have to be turned off to effectively make the switch. VICMON may also be used with expansion RAM in the Memory Expansion Board.

To start using VICMON type SYS24576 or SYS6★4Ø96, and then press the RETURN key.

The VIC screen will now display the values currently held in the 6502's registers. An example is shown in Figure 1-1. VICMON is ready to accept your commands.

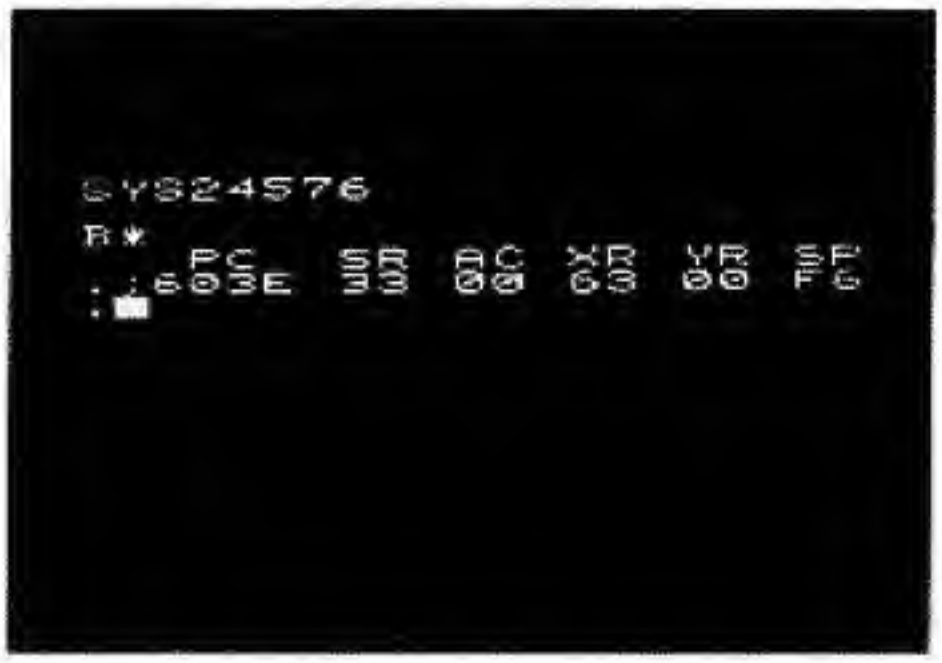

Figure 1-1 An example initial VICMON display.

1.5 Command Format

Most VICMON commands are a single alphabetic character followed by the command parameters, if required. The commands are explained in detail in Section Two. The parameters include the start address or start and end addresses, op-codes, operands, hex values, etc. The conventions and limitations for them are defined in Section 2.2.

Command statements (except for J) are terminated and execution of them is initiated by pressing the RETURN key.

A summary of the commands, their formats and where they are described is given on the back cover of this manual.

1.6 Entering Commands

Whilst the VIC is operating under VICMON you are prompted with a ".". To enter a command simply type the command letter(s) and the parameter(s). If more than one parameter is required, separate them with spaces, commas, colons or any other convenient symbol. The command (except for J) is terminated and execution is begun when the RETURN key is pressed.

1.7 Error Indication

Any errors you make when inputting the command statements are indicated by a question mark following the position of the error. You may re-type or correct the command using the standard VIC editing facilities. Press RETURN to initiate the corrected command.

SECTION TWO
THE COMMANDS OF VICMON

2.1 Introduction

In Section Two, each VICMON command is given in alphabetical order. The required format is shown and the purpose and function of the command are explained. A simple example is included which shows the command, a typical response when it is used and an explanation of the results of executing it.

In Section Three a more detailed example is shown using an actual machine code program which is then "debugged" using VICMON.

2.2 Conventions

The parameters in the command formats are represented as follows:

(addr)	a two byte hex address, e.g. 0400
(dev)	a single byte hex device number, e.g. 08
(opcode)	a valid 6502 assembly mnemonic, e.g. LDA
(operand)	a valid operand for the preceding instruction, e.g. #$01
(value)	a single byte hex value, e.g. FF
(data)	a string of literal data enclosed in quotes or hex values. Successive items are separated with commas.
(ref)	a two byte hex address, e.g. 2000
(offset)	a two byte hex offset value, e.g. 3000.

2.3 The Commands

2.3.1 A — ASSEMBLE

Format : A (addr) (opcode) (operand)

Purpose : To assemble code starting from a specified address.

The command allows you to input assembly code line by line and have it stored as machine code. When the command is entered, the appropriate code is written in memory beginning at the specified address. The address of the next available memory location beyond that required by the specified op code and operand is then prompted awaiting input of additional code.

To terminate the A command, simply press RETURN when the new address is prompted.

If you input an illegal op code or operand, VICMON will place a question mark after the illegal quantity and will return you to the monitor with a prompt (·) on a new line. If you fail to specify either the op code or operand, VICMON will ignore the line and return you to the monitor with a prompt (·) on a new line.

NOTE: All operands must be given as hex numbers preceded by a dollar sign, i.e. typed as $nn.

EXAMPLE : To enter the following machine code:

```
LDA #$19
JSR $FFD2
RTS
```

beginning at address $1000.

```
COMMAND: A 1000 LDA #$19 (RETURN)
DISPLAY  : .A 1000 LDA # $19
           .A 1002.
TYPE     : JSR $FFD2 (RETURN)
DISPLAY  : .A 1000 LDA #$19
           .A 1002 JSR $ FFD2
           .A 1005
TYPE     : RTS (RETURN)
DISPLAY  : .A1000 LDA #$19
           .A1002 JSR $ FFD2
           .A1005 RTS
           .A1006
```

RESULT : The machine code equivalent of the specified assembly language code is stored in memory from location 1000 to 1005 inclusive.

2.3.2 B — BREAKPOINT

Format : B (addr)

or: B (addr), n
where n is a four digit HEX number indicating how many times that address will be encountered before the break occurs.

Purpose : To set a breakpoint so that a program does not execute fully but instead stops at the specified location.

A breakpoint allows you to run your program up to a specified address. If you used G (see Section 2.3.6) to initiate the run, the contents of

the registers are displayed automatically, allowing you to determine if they are as expected. The Q mode of execution (see Section 2.3.13) will also stop at the breakpoint but the registers will not be displayed. You will be switched to the W mode. (See Section 2.3.18.) To display the registers, press the STOP key and then use the R command. (See Section 2.3.14.)

Note that the run terminates before the instruction in the specified address is executed. You must be careful not to set a breakpoint between an op code and its operand or in the midst of data. Doing so will cause the breakpoint to be ignored and the resulting run to be unpredictable.

It is possible to execute an address a specified number of times and break on the subsequent pass. This is done by specifying the number of iterations (the n in the format statement above). If no number is given after the address, the program will break when that address is encountered the first time.

Type in the breakpoint command and press the RETURN key. Now run the program using the G or Q command. The program will run up to the breakpoint and stop (unless the program has logical faults which prevent its reaching the specified address).

You can only set one breakpoint before you begin running your program. When a breakpoint is reached you may then set a new breakpoint if you wish. You can resume the program execution after a breakpoint using any of the run commands (G, Q, W, J), i.e. you are not restricted to using the same method of running throughout.

If a breakpoint is never reached, execution can be stopped by pressing the STOP key and then the RESTORE key. This will return you to BASIC. You should re-enter VICMON and set an earlier breakpoint to isolate your problem. (See Section Three.)

EXAMPLE : Assume that you have a program in memory from location 1000 to location 1200. To stop the program execution before 1050:

COMMAND: B 1050 (RETURN)

COMMAND: Q 1000 (RETURN)

RESULT : The program will slowly execute, stopping before line 1050 is executed. You will be left in the WALK mode.

EXAMPLE : To set a breakpoint so that the program will stop the third time location 1100 is reached:

COMMAND: B 1100, 0003 (RETURN)

COMMAND: G 1050 (RETURN)

RESULT : The program will run, stopping before location 1100 is executed the third time. The contents of the registers at that point will be displayed in the format shown in Figure 2-1.

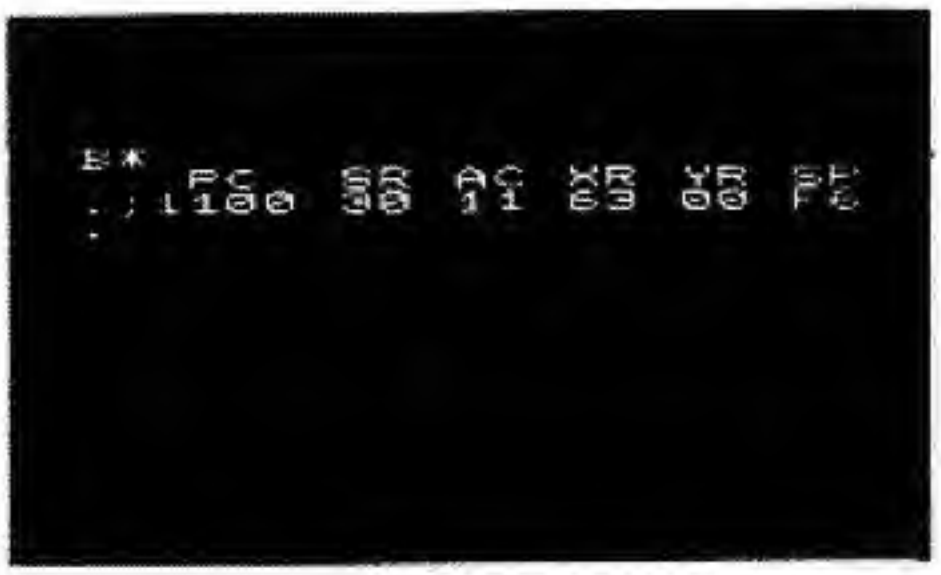

Figure 2-1 Register Display

2.3.3 D — DISASSEMBLE

Format : D (addr)
or: D (addr), (addr)

Purpose : To disassemble code from a specified point or between a range of points.

NOTE: Forward or backward scrolling with the cursor movement keys will continue to disassemble code.

The D command enables you to convert the code that is stored in the computer's memory back into assembly language notation. You may specify a beginning address in which case that line of code will be disassembled and displayed in assembly language on the screen. The VIC will remain in the disassemble mode and you may use the cursor to disassemble additional lines of code, i.e. the cursor-down key will disassemble the line(s) following the specified line and the cursor-up key will disassemble preceding lines. Note, however, that the disassembly will not begin until the cursor reaches the bottom (or top) of the screen and the scrolling begins.

WARNING: Working backwards with the cursor key may not give a perfectly accurate translation of the code.

Alternatively, you may specify a range of addresses to be disassembled. The lines specified will be displayed on the screen. If you specify a range of addresses that is too long to be displayed on the screen at one time, the screen will scroll. The STOP key will terminate the scrolling and you will remain in the disassemble

mode. You may disassemble subsequent lines with the cursor-down key.

While you are in the disassemble mode, a line of code on the screen can be modified by simply correcting or retyping the line and pressing RETURN. The A command is automatically activated. When you have made the change, you remain in the A mode with the cursor positioned after the address on the line following the corrected line. To terminate the assemble mode, clear the screen and press RETURN.

EXAMPLE : To disassemble the lines of code input in the example of the assemble command and then to change the address in the second line to FFD0:

COMMAND: D 1000, 1005 (RETURN)

DISPLAY : .1000 LDA # $19
.1002 JSR $ FFD2
.1005 RTS

ACTION : Move the cursor so that it is positioned over the 2 in the FFD2.

TYPE : 0 (RETURN)

DISPLAY : .1000 LDA # $19
.A1002 JSR $FFD0
.A1005 RTS

RESULT : The code from location 1000 to location 1005 is disassembled. The change is made and then stored with the RETURN key. You are left in the assemble mode.

2.3.4 E — ENABLE Virtual Zero Page

Format : E (addr)

Purpose : To set aside a virtual zero page so that VICMON does not interfere with your variables.

VICMON uses location $00 to $71 of zero page. The Kernal uses the rest of the zero page and some of the $200-$800 pages. Since your program may assign variables which will be stored on the zero page, running the program may interfere with some of the information already stored there. To prevent this, the E command enables you to set aside a virtual zero page of 256 bytes at another location. When a virtual page has been set and your program is run, VICMON automatically swaps the zero page contents with the virtual zero page contents, thus protecting the VICMON and Kernal information. When program execution is terminated, they are swapped again.

To disable the virtual zero page when you have your program running correctly, simply use the E command with the zero page address, i.e. E0000.

EXAMPLE : To set virtual zero page beginning at location $1000:

COMMAND: E 1000 (RETURN)

RESULT : The locations $1000 to $10FF will be set aside as a virtual zero page.

2.3.5 F — FILL Memory

Format : F (addr), (addr), (value)

Purpose : To fill memory between two specified addresses with a given value.

The F command enables you to put a known value into a specified block of memory. This is useful for initializing data structures or for blanking out the contents of any RAM area. Simply specify the range of the block of memory and the pattern you wish to write in that block. Naturally you should not specify addresses from $0000 to $01FF (pages zero and one). Similarly, if you are using a virtual zero page (see the E command, Section 2.3.4) you should avoid that area as well.

EXAMPLE : To write $EA (a no-op instruction) from location $1000 to $2000 inclusive:

COMMAND: F 1000,2000,EA (RETURN)

RESULT : The no-op instruction ($EA) is written in all the addresses from $1000 to $2000.

2.3.6 G — GO

Format : G
or: G (addr)

Purpose : To execute a program beginning at the location currently in the program counter or beginning from a specified address.

The G command may be used alone or stated with an address. When G is used alone, the VIC will execute the program in memory beginning with the location currently in the program counter. (To display the contents of the program counter, use the R command as described in Section 2.3.14.) When an address is given with the G command, execution will begin at the location specified.

The G command restores the registers to their last known states and if a virtual zero page is active (see the E command), exchanges VICMON's zero page with the virtual zero page. Execution of your program will continue until a preset breakpoint, if any (see the B command) or until the end of the program is

reached, unless the program has logical flaws. If the execution is terminated by a breakpoint, the contents of the registers at that point will be displayed. If the program is terminated by RTS, when that command is reached you will be returned to BASIC. If the last command is BRK, when it is reached you will be returned to VICMON. If no terminator is attainable due to a flaw in the program, you will have to use the STOP and RESTORE keys to terminate the execution. You will then be in BASIC and must re-enter VICMON.

NOTE: If your program has changed the screen and/or letter colours you may be unable to see the READY or the register display.

For other means of executing programs, see the J, W and Q commands, Sections 2.3.9, 18 and 13 respectively.

NOTE: Frequent breakpoints can prevent the program becoming "runaway".

EXAMPLE : Assume that you have a program in memory and wish to begin executing it from location $2000:

COMMAND: G 2000 (RETURN)

RESULT : The registers will be restored. The PC will be set to $2000. If a virtual zero page has been established, it will be swapped with the VICMON zero page. The program will begin executing at $2000.

2.3.7 H — HUNT

Format : H (addr), (addr), (data)

Purpose : To search through a specific block of memory and locate all occurences of particular data or character strings.

The H command easily locates any specified character pattern that is in the computer's memory and displays it on the screen. You may use this command to locate data, which is specified in hex, or to find text strings up to 88 characters long (one line), which are specified literally and preceded by a single quote mark. All locations within the specified range which contain the requested characters will be found. If there are more occurrences than will fit on the screen, the screen will scroll. The STOP key will terminate both the scrolling and the HUNT and return you to VICMON. The control key will slow down the rate of the scroll. When all occurences within the range have been located, you will be returned to VICMON.

EXAMPLE : Assume that the data string $A92F3C is stored in memory somewhere between location $C000 and location $ C0FF. To locate the string:

COMMAND: H C000,C0FF,A9,2F,3C (RETURN)

RESULT : Memory is searched between $C000 and $C0FF and the location where $A92F3C is stored is displayed.

EXAMPLE : Assume that the word COMMODORE is stored in memory in three locations between $2000 and $3000:

COMMAND: H 2000,3000,'COMMODORE (RETURN)

DISPLAY : See Figure 2-2.

Figure 2-2 Example of Character String Display.

2.3.8 I — INTERPRET

Format : I (addr), (addr)
or: I (addr)

Purpose : To locate and display printable text characters within a specific block of memory.

The I command will display in reverse video any of the 96 printable CBM ASCII code equivalents occuring within the specified block of memory. All other characters in the block will be indicated by a dot (.). If the specified block more than fills the screen, the screen will scroll. The STOP key will terminate the scrolling and the control key will slow down the rate of scrolling. When the specified INTERPRET is terminated, you will remain in the I mode. Pressing the cursor-down key will display any CBM ASCII characters on the next line as the screen scrolls.

EXAMPLE : Assume that the hex codes for C, carriage return, line feed, CB, carriage return, line feed, and CBM, carriage return, line feed

are stored beginning at location $1000:

COMMAND: I 1000 (RETURN)

DISPLAY : See Figure 2-3.

Figure 2-3 Display Printable Characters

2.3.9 J — JUMP to subroutine.

Format : J

Purpose : To execute a subroutine call and return without single-stepping whilst running a program under the W command.

The W command runs your program one line at a time, i.e. after executing a line, it waits for an input from you before proceeding. Whilst in this mode, you may wish to execute a sub-routine all at once, for example, when you have already checked it by single stepping. You may wish to move through the sub-routine quickly on subsequent calls. The J command enables you to do this. Simply press the J key when the line containing the subroutine call is displayed. It is not necessary to press the RETURN key. Note that the J will not appear on the screen.

When the J command is used, the correct return address is pushed on the stack and the subroutine is executed. When the final RTS instruction is encountered, the program counter will be set to the return address which was pushed on to the stack. You will again be in the WALK mode.

EXAMPLE : Assume that you are in the WALK mode and the following code is on the screen:

147F LDA #$00
1481 JSR $A2C7

COMMAND: J

RESULT : $1484 is pushed on the stack. The subroutine beginning at $A2C7 is executed. When the RTS is reached, the stack is popped to the PC. You are returned to W mode.

2.3.10 L — LOAD

Format : L "FILENAME", (dev)

Purpose : To load a program file into memory from a specified device.

The L command enables you to read a load file or a program file that is stored on cassette or on diskette and write it into the VIC's RAM. For disk files, the address of the first location in RAM into which the load file will be read must be the first two bytes of the file. Tape files have the start address as part of the initial header block.

NOTE: Only program files that have been created using the S command of VICMON (see Section 2.3.16) or SAVE in VIC BASIC may be loaded with this option.

The command consists of L, the name of the file and the number of the device to read from. The file name must be enclosed in quotation marks and may be any legal VIC file name. The device number of the cassette unit is 01. The device number of the disk unit is 08.

When the L command is used, the specified file on the device will be read into memory until an EOF is encountered. If the EOF is not encountered, the LOAD will not terminate and you will have to press the STOP and RESTORE keys to stop it.

If the device or file is not present you will get an error message and be returned to BASIC.

EXAMPLE : Assume that you have a disk program file named TEST that is 258 bytes long, the first two bytes of which are 00CA. To read this file into memory:

COMMAND: L "TEST", 08 (RETURN)

RESULT : The program named TEST which is on the diskette in the disk unit is loaded into memory from CA00 to CB00 inclusively.

2.3.11 M — MEMORY

Format : M (addr), (addr)

or: M (addr)

Purpose : To display the hex code that is stored in a given block of memory.

The M command will display the contents of memory from the beginning address in the command up to and including the contents of the ending address. The display will have the address and five hex bytes on a line. If only one address is given in the command, five bytes will be displayed beginning with the contents of the specified address.

Additional groups of five bytes may be displayed by causing the screen to scroll, i.e.

using the cursor control keys. Note that if you specify a second address which is smaller than the first you will wrap around from the end of memory to the beginning.

The contents of memory may be changed by typing over the displayed values and then pressing the RETURN key. If there is a bad RAM location or if you attempt to modify ROM, a ? will be displayed at the location of the impossible change.

EXAMPLE : To display five bytes of memory beginning at location $1000 and to change the 00 to FF:

COMMAND: M 1000 (RETURN)

DISPLAY : .1000 A0 00 EA EA FF

ACTION : Position the cursor over the first 0 of 00. Type FF and press RETURN.

RESULT : The five bytes of the memory beginning at location $1000 now read A0 FF EA EA FF.

2.3.12 N—NUMBER

Format : N (addr),(addr),(offset), (lowlim),(uplim),W

where offset is a hex value indicating the amount to be added to the existing addresses and lowlim and uplim specify the range of the operands to be offset and W is an optional command indicating that the range is a word table.

Purpose : To reassign absolute memory addresses between specified ranges when a program has been relocated with the T command.

With the T command (see Section 2.3.17), you can relocate your program to another part of memory. Of course, if your program contains absolute addresses, these addresses will no longer be valid. The N command allows you to automatically change these values. First you must calculate the amount you have moved the program. Note that if you have moved the program to a lower memory location, you must calculate the wrap-around value, e.g. if your program was at $A000 and is moved to $0400, you have moved $6400 since $A000 + $6400 = $10400. The value $6400 is the offset.

With the N command you may change all absolute addresses or only those within a specific range. The range is established by setting upper and lower inclusive limits. You must also specify the block of memory in which the change is required. VICMON will take each operand within the block and add the amount of the offset to it, i.e. it will overlook three bytes and add the offset to the next two bytes. Of course, if you want to change a word table, this would be disastrous, but VICMON has provided for this with the optional W at the end of the N command. When the W is included, every word, i.e. every two bytes, will be offset rather than as described above.

WARNING: Do not use the N command in the range of your data locations or you will destroy the data's usefulness.

EXAMPLE : Assume that you have used the TRANSFER command to relocate your program. It was in locations $1000 to $2000 and now is at $1500 to $2500. To appropriately adjust all the absolute addresses in that range:

COMMAND: N 1500,2500,0500,1000,2000 (RETURN)

RESULT : Within the code in locations $1500 to $2500, all absolute addresses that fall between $1000 to $2000 are increased by $500.

2.3.13 Q—QUICK TRACE

Format : Q

or: Q (addr)

Purpose : To run a program at a slow pace beginning at the specified address and checking for a breakpoint or your use of the STOP and X keys after each instruction is executed.

The Q command, like the G command (see Section 2.3.6), may be used alone or stated with an address. When it is used alone, VICMON will execute the program in memory beginning with the location currently in the program counter. (To display the contents of the program counter, use the R command as described in Section 2.3.14.) When an address is given with the Q, execution will begin at the location specified.

The Q command functions much as the G command with one major exception. Whilst G turns program control completely over to the CPU, Q executes one instruction at a time, checking after each step to see if a breakpoint is set or if you have asked for execution to terminate. This breakpoint check allows you to set breakpoints in ROM as well as in RAM. When the breakpoint is reached, execution will stop and you will be in the WALK mode. (See Section 2.3.18.) To display the registers at this point, press STOP, R and RETURN.

The user interrupt can be generated from the keyboard at any point. Simply press the STOP key and then the X key. Execution will be

terminated and the contents of the registers at that point will be displayed.

EXAMPLE : To execute a program in QUICK TRACE mode beginning at $1000:

COMMAND: Q 1000 (RETURN)

RESULT : The PC is set to 1000. The registers are initialized. If a virtual zero page (see the E command, Section 2.3.4) has been established, it is swapped with the zero page. Program execution is begun at line 1000.

2.3.14 R — REGISTERS

Format : R

Purpose : To display the contents of the registers.

The R command enables you to view the current status of the following registers in the VIC 20's 6502:

program counter	PC
status register	SR
accumulator	AC
index register X	XR
index register Y	YR
stack pointer	SP

This can be useful when you are debugging a program because the R enables you to see if the registers contain the values you expected. You may also change the values in the registers whilst in the R mode by simply typing over a new value and pressing RETURN. The register display is automatically generated when VICMON is started up, when a preset breakpoint (see Section 2.3.2) is reached in the G mode (see Section 2.3.6), and when a Q run (see Section 2.3.13) is terminated by the STOP and X key combination.

EXAMPLE : To display the contents of the registers:

COMMAND: R (RETURN)

RESULT : Figure 2-4, for example.

Figure 2-4 An example of Register Displays

2.3.15 RB — REMOVE BREAKPOINT

Format : RB

Purpose : To remove a breakpoint.

Breakpoints are set by the B command (see Section 2.3.2) and can be removed by the RB command. Simply specify RB and the breakpoint which was set will be removed. If no breakpoints exist when an RB is executed, VICMON will interpret the command as if it were an R and display the registers.

EXAMPLE : Assume that a breakpoint was set at location $1050. To remove that breakpoint:

COMMAND: RB 1050 (RETURN)

RESULT : A breakpoint no longer exists at location $1050.

2.3.16 S — SAVE

Format : S "filename",(dev),(addr),(addr)

Purpose : To write the contents of a specified RAM area to a particular device.

The S command enables you to save a program on diskette or cassette so that it can be used at a later time. The command consists of the name of the file, the number of the device to be written to and the start and end address of the RAM block. The file name must be enclosed in quotation marks and must obey the syntax rules for VIC files, i.e. it must begin with an alphabetical character and be no more than 16 characters long. The device number of the cassette unit is 01 and of the disk unit, 08. The final address must be one larger than the location of the last byte you wish to write.

WARNING: If the final address is not one larger than the location of the last byte you wish to save, the last byte will be lost.

If the specified device is not present you will get an error message and be returned to BASIC.

NOTE: VICMON will not save memory above $7FFF, i.e. the start address must be $0000 or greater but not larger than $7FFF and the end address must be greater than $0000 but no larger than $8000. If you attempt to save memory outside this range, only the file header will be saved, i.e. no data or program will be written.

EXAMPLE : Assume that you have a program in memory from location $1000 to $10FF. To write that program to the diskette in the disk drive, naming that program TEST 1:

COMMAND: S "TEST1",08,1000,1100 (RETURN)

RESULT : A file named TEST 1 will be written on the diskette. It will

contain the code that was in RAM location $1000 to $10FF inclusive.

2.3.17 T — TRANSFER

Format : T (addr),(addr),(addr)

Purpose : To transfer the contents of a block of memory from one area of RAM to another.

The T command enables you to relocate your program or data to another part of the memory. This can be useful if you wish to expand a program or to use part of a program elsewhere without retyping. The command consists of three addresses. The first two indicate the block of memory to be duplicated. The third address indicates the starting address for the copy.

If a program is transferred and the program contains absolute addresses or word tables, these specifications in the new location will not be accurate. The N command (see Section 2.3.11) allows you to offset these values by the appropriate amount so that the relocated program will run properly.

EXAMPLE : Assume that you have a block of data in memory from location $3000 to $3500. To move that data to a new location beginning at $4000:

COMMAND: T 3000,3500,4000 (RETURN)

RESULT : The data is now in the block $3000 to $3500 and in the block $4000 to $4500.

2.3.18 W — WALK

Format : W

or: W (addr)

Purpose : To execute a program one instruction at a time.

The W command executes the line of code indicated by the address in the program counter, if W is used alone. Alternatively you may specify the address of the instruction to be executed.

When using W, the first instruction is executed and the second instruction will appear on the screen. VICMON will wait for you to press the space bar before it will execute the second line. When the space bar is pressed, the line will be executed and the next line displayed. In this way you can WALK, i.e. single-step through the program. To return to VICMON from W, press the STOP key.

You may use the R command (see Section 2.3.14) to display the contents of the registers at any point. Press STOP, then the R key and then RETURN to accomplish this.

Each subroutine must be single-stepped as well, unless you use the J command to treat the entire sub-routine as one step (see Section 2.3.9).

EXAMPLE : To single step through a program beginning at location $1000:

COMMAND: W 1000 (RETURN)

RESULT : The instruction stored at address $1000 is executed and the next instruction is displayed.

ACTION : Press the SPACE BAR.

RESULT : The second instruction is executed and the third instruction is displayed.

2.3.19 X — EXIT to BASIC

Format : X

Purpose : To terminate VICMON control and return to BASIC.

Use of the X command returns you to BASIC. Your program will remain in memory but any breakpoint or virtual zero page assignments will not be preserved.

EXAMPLE : To exit VICMON:

COMMAND: X (RETURN)

RESULT : You will be returned to BASIC and prompted with READY.

SECTION THREE

USING VICMON AS A DEBUGGING TOOL

1.1 Introduction

The following is an example which shows some of the editing and fault tracing facilities of VICMON. It uses a 6502 assembly language program and VICMON to show how an error is located in the program and fixed. More details on the individual commands used here are given in Section Two.

If you wish to try the example, follow the instructions beginning in Section 3.3. If not, it is suggested that you at least read through the example.

3.2 The Example Program

The program used in this section writes a screen full of each of the printable characters in turn. Two screen positions are left blank to prevent the screen from scrolling. A flowchart of the program is shown in Figure 3-1.

The program uses the ROM routine $FFD2 to print a character. First, the screen is cleared by the following commands:

```
LDA #$93
JSR $FFD2
```

Then a loop fills all but the last two character positions on the screen with spaces. There are 506 character locations possible on the screen, so 504 have to be filled. This is equivalent to two lots of 252 ($FC).

Once this has been done, an indirect pointer to the screen is set up in zero page, using the contents of $0288 to point to the start of the screen. Two is added to this to reference the end of screen for testing. Using a loop, the screen (all positions filled with spaces) are filled with the first character (value 0) then the second and so on until all 256 characters have been in each position on the screen where there had been a space.

It is necessary to print to the screen to ensure that characters appear when they are stored directly to the screen area (STA ($01),Y). It is possible to store values in the colour RAM area of memory instead but this requires the use of an additional indirect pointer.

The page number of the screen is stored in

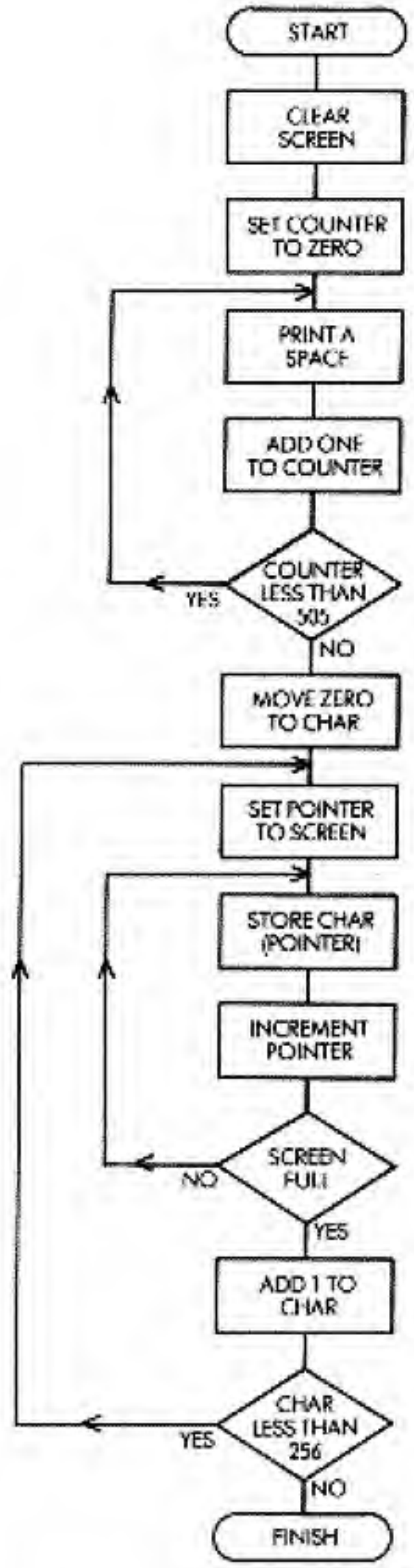

Figure 3-1 Flowchart of Example Program

location $0288 which means the screen starts at $($0288)00. This method is required because if you use expansion memory, the location of the screen RAM alters.

3.3 The Procedure

3.3.1 INPUTTING THE PROGRAM

These are the steps to input the program described above and to locate a fault in it. Insert the VICMON cartridge into the VIC or VIC expansion board. Switch on the computer (and the expansion board if you are using one). Then type SYS (6*4096) to start VICMON. Next using the A command (see Section 2.3.1) type in the following code:

```
1000 LDA# $93
1002 JSR $FFD2
1005 LDY #$00
1007 LDX #$00
1009 LDA# $20
100B JSR $FFD2
100E INX
100F CPX #$FC
1011 BNE $1009
1013 INY
1014 CPY # $02
1016 BMI $1009
1018 LDX # $00
101A STX $01
101C LDA $0288
101F STA $02
1021 CLC
1022 ADC # $02
1024 STA $00
1026 LDY #$00
1028 TXA
1029 STA ($01),Y
102B INY
102C BNE $1029
102E INC $02
1030 LDA $02
1032 CMP $00
1034 BNE $1029
1036 INX
1037 BNE $101C
1039 BRK
103A BRK
```

Once you have entered the program save it on the cassette unit (see Section 2.3.16) with the following command:

S "PROGRAM",01,1000,103A (RETURN)

(To save to diskette, substitute 08 for 01 in the above command.)

This is a safeguard so that you do not have to type the program in again, if for example, power to the computer is lost.

Assuming that the program will work first time (a rare occurrence with machine code programs), use the GO command (see Section 2.3.6) and type:

G 1000(RETURN)

If you have typed in the program exactly as listed above, the top half of the screen will display a series of characters very rapidly. The bottom 3 lines of the screen will be blank. After a very short time the program will finish and the screen will appear as shown in Figure 3-2.

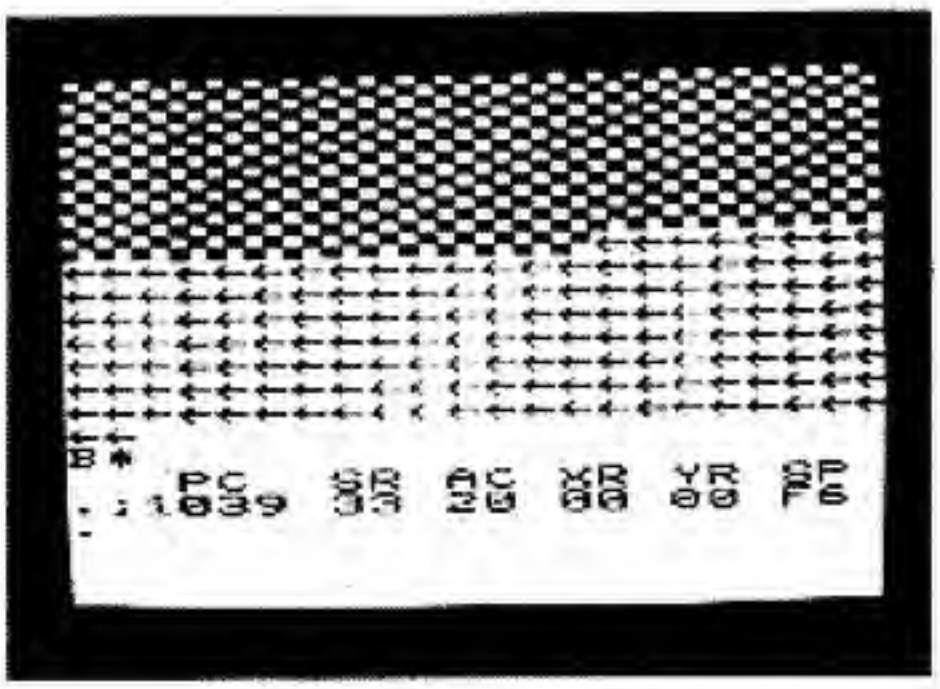

Figure 3-2 Result of first attempt to run example program

3.3.2 LOCATING THE FAULT

Obviously something has gone wrong and you must locate the problem. Here is a typical technique. First, split the program up into two sections, the first of which will clear the screen and fill it with spaces. This section ends at $1018, so set a breakpoint (see Section 2.3.2) at $1018 by typing:

B 1018 (RETURN)

So that you can see what is happening, change the character printed from a space ($20) to an A ($41). Do this with:

A 1009 LDA # $41 (RETURN) (RETURN)

To slow the operation down, use the quick trace option (see Section 2.3.13):

Q 1000 (RETURN)

This executes the program at a pace much slower than normal.

If you are using zero page locations for your program, it is advisable (and usually necessary) to make use of the virtual zero page option (see Section 2.3.4) because VICMON uses the zero page and your program and VICMON may overwrite each other. This option is not required

in this first section, but will be required in the second.

Since enabling the virtual zero page before the quick trace is executed will result in taking approximately 2 minutes to clear the screen and at least twice that time to fill it with characters afterwards, do not enable it now.

As the section of code ($1000-$1018) executes, you will notice that rather than stopping 3 lines short of the end of the screen, the characters overflow the end of the screen. The screen scrolls up (4 lines) and then an extra 2 characters are printed. This means that 4 unwanted characters are being printed. The most likely cause of this is that the test for the number of characters printed is being performed incorrectly. The value in the X register should run between $00 and $FC while Y is 0 and Y is 1. If you look carefully you will notice that once the value of $FC is reached in X, the value of Y is increased. If it is less than 2, a space ("A") is loaded into the accumulator and is printed. This means that X goes $00-$FF, $00-$FC giving the four extra characters. To fix this, the branch at $1016 must be altered to point to $1007 instead of $1009. Since quick trace leaves you in walk mode, you must press STOP to return to VICMON. Next, type:

A 1016 BMI $1007 (RETURN) (RETURN)

If you now type:

Q 1000 (RETURN)

the routine will stop at the correct point on the screen and appear as in Figure 3-3.

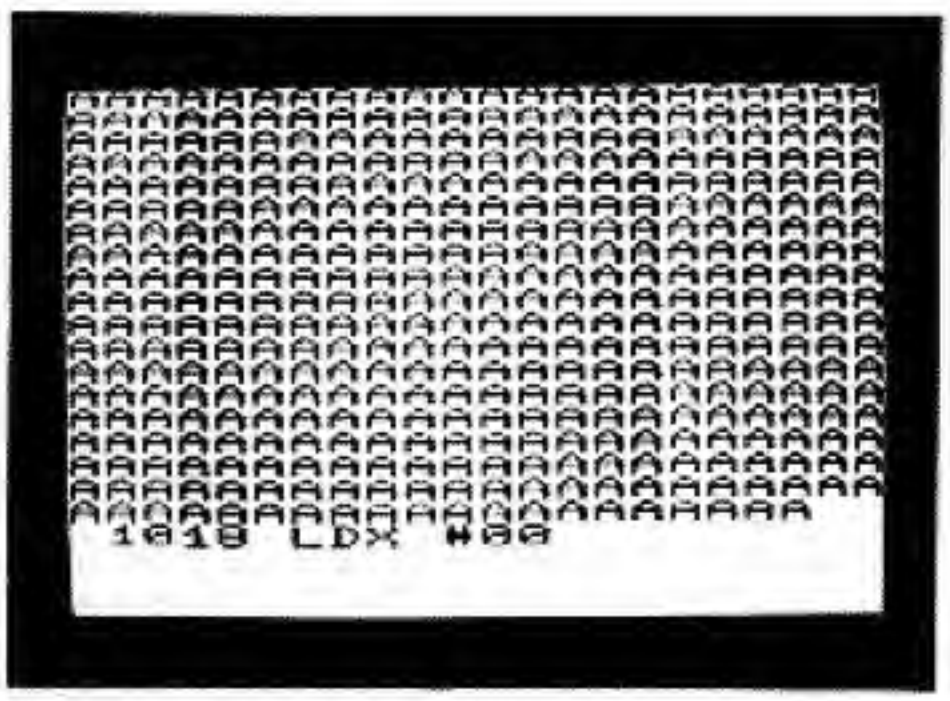

Figure 3-3 Screen filled with A's

At this point you can see that the first part of the routine worked fine. Now you should enable the virtual zero page. Press STOP, then type:

E 1800 (RETURN)

This will assign the virtual zero page to a block of memory starting at location $1800. Now execute the code at a slow rate starting where it finished before, i.e. $1018 as indicated by the PC. The QUICK TRACE mode will allow you to go slowly, stopping execution if you need to, so type:

Q (RETURN)

The A's on the screen will start turning into @'s until about half way down the screen (257th character). The remaining A's will be replaced by left arrow symbols. Then the @'s will start turning back to A's. However, when the last @ changes, nothing will happen to the screen for a short time and then the A's will begin turning into B's. The left arrow symbols will remain. Note that the screen character for 0 is @ , 1 is A, 2 is B, etc.

Press STOP and X together to interrupt the program, because, as you can see, there is still a problem.

Use the WALK command (see Section 2.3.18) to single step through the program to see if you can spot where the wrong character is coming from. Type:

W 1018 (RETURN)

After a period of time (the interval depends upon whether you press the SPACE BAR or hold it down) the following will appear on the screen:

```
1010 2E INC 02
1030 LDA 02
1032 CMP 00
1034 BNE 1029
1029 STA (01),Y
```

At this point the accumulator contains the high byte of the screen pointer (for the second half of the screen). This shows up as a left arrow on the screen. If you terminate the WALK mode (press the STOP key) and display the registers by typing R (RETURN) (see Section 2.3.13), you will see:

```
.R
 PC  SR  AC XR  YR  SP
102B A0  1F 00  00  F2
```

The values of PC, SR, YR may vary depending on when you pressed the STOP key, i.e. which instruction is to be performed next.

At this stage of the program, the accumulator should contain the same value as the X register. As the high byte of the screen pointer is loaded into the accumulator, it is necessary to transfer the value from the X register into the accumulator. The instruction for that is TXA which is at $1028, so the branch to $1029 must be changed. Type:

A 1034 BNE $1028 (RETURN) (RETURN)

Note that the branch at $102C does not need to be altered because the value of the accumulator is not altered within this inner loop.

Remove the breakpoint, restore the spaces in the initial step, and save the program again, by typing:

RB (RETURN)

A 1009 LDA#$20 (RETURN) (RETURN)

S "PROGRAM",01,1000,103A (RETURN)

G 1000 (RETURN)

It should now work. If it does not, compare it with the following program, by using the D command to display the stored code. Spot the difference, and make any necessary alterations by using the A command. Remember, when writing your own programs or routines you would not have a correct copy of the program to compare with. However, the same principles of setting breakpoints, executing slowly and checking registers applies for any program.

```
.,1000 LDA #$93
.,1002 JSR $FFD2
.,1005 LDY #$00
.,1007 LDX #$00
.,1009 LDA #$20
.,100B JSR $FFD2
.,100E INX
.,100F CPX #$FC
.,1011 BNE $1009
.,1013 INY
.,1014 CPY #$02
.,1016 BMI $1007
.,1018 LDX #$00
.,101A STX $01
.,101C LDA $0288
.,101F STA $02
.,1021 CLC
.,1022 ADC #$02
.,1024 STA $00
.,1026 LDY #$00
.,1028 TXA
.,1029 STA ($01),Y
.,102B INY
.,102C BNE $1029
.,102E INC $02
.,1030 LDA $02
.,1032 CMP $00
.,1034 BNE $1028
.,1036 INX
.,1037 BNE $101C
.,1039 BRK
.,103A ???
.
```

3.4 Summary

Here is a summary of the steps to follow to use VICMON to debug your own programs:

1. Initialize VICMON.
2. Load your program with L
   or type it in and SAVE it.
3. Attempt to run the program from the start address using G.
4. Set breakpoints to determine area of fault.
5. Disassemble faulty section (or all of program, if it is short) using D. It is preferable to list the disassembled code on your printer.
6. Quick trace through the faulty section, especially if it involves screen displays.
7. Walk through the faulty section.
8. Display registers at various points to check values, if necessary. Use M and I to display areas of data or working variables.

9a. Use A to correct faulty code.

b. Use M to correct faulty data.

10. Keep a note of changes made. Save the program frequently.
11. Remember that you may need to use virtual zero page (E) while you are using the quick trace and walk options.
12. If you cannot find the problem, go back to your flowchart(s) and re-think your logic.

INDEX

SUMMARY OF COMMANDS

All commands except J are terminated and execution is begun by pressing the RETURN key.

The parameters in the command formats are represented as follows:

(addr)	a two byte hex address, e.g. 0400
(dev)	a single byte hex device number, e.g. 08
(opcode)	a valid 6502 assembly mneumonic, e.g. LDA
(operand)	a valid operand for the preceding instruction, e.g. #$01
(value)	a single byte hex value, e.g. FF
(data)	a string of literal data enclosed in quotes or hex values. Successive items are separated with commas.
(ref)	a two byte hex address, e.g. 2000
(offset)	a two byte hex offset value, e.g. 3000

To start monitor, type SYS24576 or SYS64096

commodore COMPUTER